Neural architectures optimization and Genetic algorithms

MOHAMED ETTAOUIL and YOUSSEF GHANOU
UFR : Scientific calculation and Computing, Engineering sciences
Department of Mathematics and Computer science
Faculty of Science and Technology of Fez
Box 2202, University Sidi Mohammed ben Abdellah Fez
MOROCCO
mohamedettaouil@yahoo.fr yghanou2000@yahoo.fr

Abstract: The artificial neural networks (ANN) have proven their efficiency in several applications: pattern recognition, voice and classification problems. The training stage is very important in the ANN's performance. The selection of the architecture of a neural network suitable to solve a given problem is one of the most important aspects of neural network research. The choice of the hidden layers number and the values of weights has a large impact on the convergence of the training algorithm. In this paper we propose a mathematical formulation in order to determine the optimal number of hidden layers and good values of weights. To solve this problem, we use genetic algorithms. The numerical results assess the effectiveness of the theorical results shown in this paper and computational experiments are presented, and the advantages of the new modelling.

Key-Words: Artificial neural networks (ANN), Non-linear optimization, Genetic algorithms, Supervised Training, Feed forward neural network.

1. Introduction

In recent years, neural networks have attracted considerable attention as they proved to be essential in applications such as content-addressable memory, pattern recognition and optimization [10], [7]. Learning or training of ANN is equivalent to finding the values of all weights such that the desired output is generated to corresponding input, it can be viewed as the minimization of error function computed by the difference between the output of the network and the desired output of a training observations set [3].

In most instances for neural networks, multilayer neural networks that are trained with the back propagation algorithm have been used. The major shortcoming of this approach is that the knowledge contained in the trained networks is difficult to interpret [15] and [3]. Error Back propagation algorithm for neural networks is based on gradient descending technique. It suffers from problems of iterative computing for training. We can use evolutionary algorithms which realize a global search [6].

Global search may stop the convergence to a non-optimal solution and determine the optimum number of ANN hidden layers. Recently, some studies in the optimization architecture problems have been introduced [8], in order to determine neural networks parameters, but not optimally.

Traditional algorithms fix the neural network architecture before learning [19], others studies propose constructive learning [22], [23], it begins with a minimal structure of hidden layer, these researchers initialised the hidden layer, with a minimal number of hidden layer neurons. The most of researchers treat the construction of neural architecture (structure) without finding the optimal neural architecture [15].

In this paper, we propose a new modelling of this problem as a mathematical programming. We apply genetic algorithms to find the optimal number of hidden layers, the activation function and the matrix of weights. We formulate this problem as a non linear programming with mixed constraints. Genetic algorithm is proposed to solve it. Consequently we determine the optimal architecture and we can adjust the matrix of weights (learning or training the artificial neural network). In our approach, we assign to each hidden layer a binary variable witch takes the value 1 if the layer is activated and 0 otherwise.

This paper is organized as follows: Section 2 describes the artificial neural networks. In Section 3, we present the problem of optimization neural architecture and a new modelling is proposed. Section 4 shows how we apply a simple genetic algorithm to solve this problem. Section 5 contains illustrative simulation results.

2. Artificial neural networks architecture

Artificial neural network (ANN) is a computing paradigm designed to imitate the human brain and nervous systems, which are primarily made up of neurons [4].

2.1. Multilayer network

The neural multilayer network (MLN) was first introduced by M. Minsky and S. Papert in 1969 [4]. It has one ore more hidden neuron layers between its input and output layers (Fig1), where there is no connection between neurons in the same layer. Usually, each neuron is connected to all neurons in the next layer.

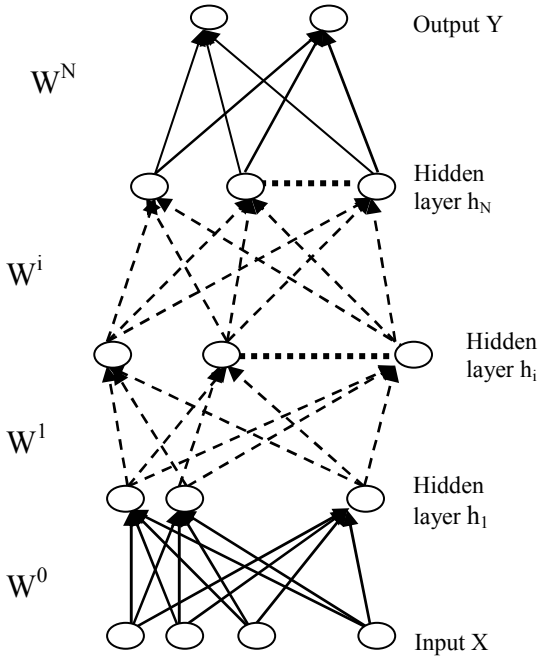

Fig1: Neural network structure

The Figure 1 shows a feed-forward neural network with N hidden layers of neurons. Note that each neuron of a certain layer is connected to each neuron of the next layer.

Determining the parameters of the ANN has a large impact on the performance of ANN, it must be considered carefully. Parameters of the ANN are network architecture, training algorithm and activation function [7].

Definitions

The term "network architecture" refers to the number of layers in the ANN and the number of neurons in each layer. In general, it consists of an input layer, one or more hidden layers and one output layer. The number of neurons in the input layer and the output layer are determined by the numbers of input and output parameters, respectively.

In order to find the optimal architecture, number of neurons in the hidden layer has to be determined (this number will be determined based on the ANN during the training process by taking into consideration the convergence rate, mapping accuracy, etc.).

Artificial neural network modelling is essentially a black box operation linking input to output data using a particular set of nonlinear basis functions. ANN consists of simple synchronous processing elements, which are inspired by biological nervous systems and the basic unit in the ANN is the neuron. ANNs are trained using a large number of input data with corresponding output data (input/output pairs) obtained from actual measurement so that a particular set of inputs produces, as nearly as possible, a specific set of target outputs.

Training MLN in a supervised method, many studies have shown MLN ability to solve complex and diverse problems.

2.2. Supervised learning

Training of ANN consists of adjusting the weight associated with each connection (weights) between neurons until the computed outputs for each set of data inputs are as close as possible to the experimental data outputs.

In other words, the error between the output and the desired output is minimized, possibly to zero. The error minimization process requires a special circuit known as supervisor; hence, the name, 'supervised learning'. It is well known that during the design and raining of ANNs, factors such as: Architecture of the ANN; Training algorithm and Transfer function need to be considered eventually.

Said supervised training is characterized by the presence of pairs, Q input and output that we will $((p_1, d_1), (p_2, d_2). \ . \ . \ (p_Q, d_Q))$, which means a stimulus p_i (input) and d_i target for this desired outputs [4].

Neurons of successive layers are interconnected, each neuron calculates the weighted sum of its inputs and output calculates this amount modified by an activation function, for example, the sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}}$$

Each input vector is passed forward through the network and an output vector is calculated. During training, the network's output is compared to the actual observations, and an error term is created. This error term is called cost function. The cost function is an important concept in training, as it is a measure of how far away, we are from an optimal solution to the problem that we want to solve. Training algorithms search through the solution space in order to find a function that has the smallest possible cost.
After training, the network reproduced the good output not only for the points of the learning base, but also for other inputs, that is known as generalization.
The learning problem can be considered as an optimization problem, where it is to identify and minimize the error between the calculated output and the desired output.
The multi-layered feed forward network trained usually with the back-propagation learning algorithm. The back-propagation learning algorithm is based on the selection of a suitable error function, whose values are determined by the actual and predicted outputs of the network.

2.3 Input layer

We suppose that $B=\{x_1,x_2,\ldots,x_Q\}$ the training base. Each observation x_i is presented at input layer of artificial neural networks.
In the training process, observations are sequentially input to the learning system in an arbitrary order.

2.4 Hidden layers

Consider a set of hidden layers (h_1, h_2, …., h_N). Assuming n_i the neurons number by each hidden layer h_i.
The outputs h_i^j of neurons in the hidden layers are computing as flows:

$$h_i^j = f\left(\sum_{k=1}^{n_{i-1}} w_{k,j}^{i-1}\ h_{i-1}^k\right),\ i=2,\ldots,N \text{ and } j=1,\ldots,n_i$$

For the first hidden layer is given as follows:

$$h_1^j = f\left(\sum_{k=1}^{n_{i-1}} w_{k,j}^0\ x_k\right),\quad j=1,\ldots,n_1$$

Where $w_{k,j}^i$ is the weight between the neuron k in the hidden layer i and the neuron j in the hidden layer i+1 , n_i is the number of the neurons in the i^{th} hidden layer, The output of the i^{th} can be formulated as following:

$$h_i = {}^t(h_i^1, h_i^2, \ldots, h_i^{ni})$$

2.6 Output layer

The output layer has neurons to collect the outputs from the hidden layer h_N..
The neurons in the output layers are computing as flows:

$$y_i = f\left(\sum_{k=1}^{n_N} w^N{}_{k,j}\ h_N^k\right)$$

$$Y = (y_1,\ldots,y_j,\ldots,y_m) = F(W,X)$$

Where $w^N{}_{k,j}$ is the weight between the neuron k in the N^{th} hidden layer and the neuron j in the output layer, n_N is the number of the neurons in the N^{th} hidden layer, $Y=(y_1,y_2\ldots,y_m)$ is the vector of output layer, F is the transfer function and W is the matrix of weights, it's defined as follows:

$$W = \left[W^0,\ldots,W^i,\ldots,W^N\right]$$

where $W^i = \left(w_{k,j}^i\right)_{\substack{1\le k\le n_i \\ 1\le j\le n_{i+1}}} \quad i=0,\ldots,N$

and

$$W = \left(w^i{}_{k,j}\right)_{\substack{0\le i\le N \\ 1\le k\le n_i \\ 1\le j\le n_{i+1}}}$$

Remark
To simplify we can take $n = n_i \quad \forall i = 1, \ldots, N$ for all hidden layers. where X is the input of neural network and f is the activation function and W^i is the matrix of weights between the i^{th} hidden layer and the $(i+1)^{th}$ hidden layer for i=1,…,N-1, W^0 is the matrix of weights between the input layer and the first hidden layer, and W^N is the matrix of weights between the N^{th} hidden layer and the output layer.

3. Optimization of Artificial Neural architecture

In this section, general information about our approach is explained.

Definition
The problem of neural architectures optimization is to find the optimal number of hidden layers in the ANN, the number of neurons within each layer, and the good activation function, in order to maximize the performance of artificial neural networks [22].

3.1 Problem formulation

The good performance is obtains by minimizing the error (distance) between the desired output and the calculated output. Determining the parameters has an impact on the performance of ANN.

The training was done in a supervised learning that is for every input vector, the desired output is given and weights are adapted to minimize an error function that measures the discrepancy between the desired output and the output computed by the network. The mean squared error (MSE) is employed as the error function.

In this article, a new optimization model is introduced, in order:

- To optimize the artificial neural architecture,
- To adjust the matrix of weights (learning)

In this new proposed strategy, we can modelize the problem of neural architecture optimization as non linear constraint programming with mixed variables. We apply genetic algorithm to solve it.

3.2 Modelization of neural architecture optimization problem

For modelling, the problem of neural architecture optimization , we have needed to define some parameters as follows:

Notation:

- N : The maximum number of hidden Layers.
- n_0 : The number of neurons by in input.
- n_{N+1} : The number of neurons by in output.
- n_i : The al number of neurons by the i^{th} hidden layer i=1,…,N.
- n_{opt} : The optimal number of hidden layers.
- X : The input of ANN
- h_i : The output of the hidden layer for *i=1,...,N.*
- Y : The calculated (actual) output of ANN
- d : The desired output of ANN
- f : The activation function of the all neurons.
- F : The transfer function of artificial neural network.
- u_i : The binary variable for *i=1,...,N*-1 and $U = (u_1,...,u_i,...,u_{N-1})$

The output of neural networks is computed by this expression:

$$F(U,W,X) = Y = (y_1, y_2,..., y_{n_{N+1}})$$

The output of first hidden layer is defined by the following expression:

$$h_1 = \begin{pmatrix} h_1^1 \\ h_1^j \\ h_1^{n1} \end{pmatrix} = \begin{pmatrix} f\left(\sum_{k=1}^{n_0} w_{k,1}^0 \; x_k\right) \\ f\left(\sum_{k=1}^{n_0} w_{k,j}^0 \; x_k\right) \\ f\left(\sum_{k=1}^{n_0} w_{k,n_1}^0 \; x_k\right) \end{pmatrix}$$

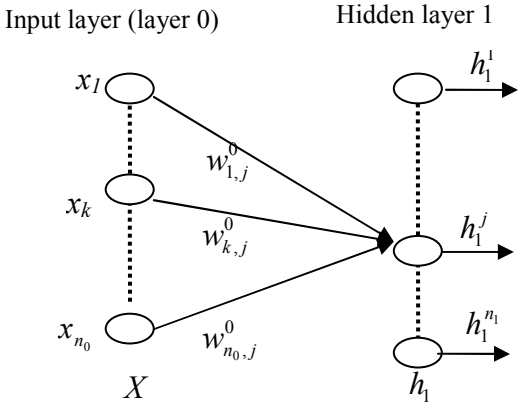

Fig 2: the output of the first hidden layer

where $(x_1, x_2,..., x_{n_0})$ is the input of neural networks.

The following term give the calculated output by the hidden layer i

$$h_i = (1-u_{i-1})h_{i-1} + u_{i-1}\begin{pmatrix} f\left(\sum_{k=1}^{n_{i-1}} w_{k,1}^{i-1} \; h_{i-1}^k\right) \\ f\left(\sum_{k=1}^{n_{i-1}} w_{k,j}^{i-1} \; h_{i-1}^k\right) \\ f\left(\sum_{k=1}^{n_{i-1}} w_{k,n_i}^{i-1} \; h_{i-1}^k\right) \end{pmatrix}$$

where $i = 2,.., N-1$

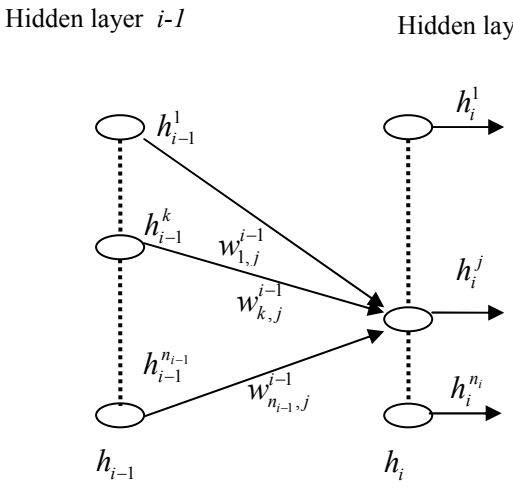

Fig 3: the output of the i^{th} hidden layer

The last hidden layer is calculated as flows:

$$h_N = (1-u_{N-1})h_{N-1} + u_{N-1}\begin{pmatrix} f\left(\sum_{k=1}^{n_{N-1}} w_{k,1}^{N-1}\ h_{N-1}^k\right) \\ f\left(\sum_{k=1}^{n_{N-1}} w_{k,j}^{N-1}\ h_{N-1}^k\right) \\ f\left(\sum_{k=1}^{n_{N-1}} w_{k,n_N}^{N-1}\ h_{N-1}^k\right) \end{pmatrix}$$

$$h_N = \left(h_N^1,..., h_N^k,...h_N^{n_N}\right)$$

The output of the neural network is calculated as flows:

$$y_j = f\left(\sum_{k=1}^{n_N} w_{k,j}^N h_N^k\right) \quad j = 1,..., n_{N+1}$$

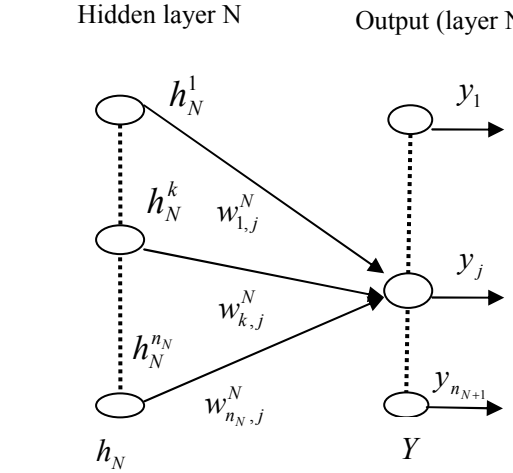

Fig 4: the output of ANN

Objective function:

The objective function of the mathematical programming model is the error between the calculated output and desired output:

$$\left\|F(U,X,W)-d\right\|^2$$

Constraints:

The last constraints guarantee the existence of the hidden layers

$$\sum_{i=1,..,N-1} u_i \geq 1$$

We will know that we will always have the output layer and one of hidden layer, so we consider duress following.

$$W = \left(w_{k,j}^i\right)_{\substack{0\leq i\leq N \\ 1\leq k\leq n_i \\ 1\leq j\leq n_{i+1}}} \quad , w_{k,j}^i \in IR$$

Where $i = 0,..., N$

If $u_i = 0$ then the ith hidden layer is deleted, and the output of (i+1)th hidden layer is computed by used the neurons of the (i – 1)th hidden layer. And we have $h_i = h_{i-1}$.

Otherwise $u_i = 1$ then the ith layer must be retained and we will: $h_i = f(W^{i-1}h_{i-1})$

The non linear programming model

The neural architecture optimization problem can be formulated as the following model:

$$(P)\begin{cases} Min\left\|F(U,W,X)-d\right\|^2 \\ Sc: \\ \sum_{i=1}^{N-1} u_i \geq 1 \\ W = \left(w_{k,j}^i\right)_{\substack{0\leq i\leq N \\ 1\leq k\leq n_i \\ 1\leq j\leq n_{i+1}}} \quad w_{k,j}^i \in IR \\ u_i \in \{0,1\} \quad i = 1,...,N-1 \end{cases}$$

where

$$U = (u_1,...,u_{N-1}) \in \{0,1\}^{N-1}$$

$W = \left(w_{k,j}^{i}\right)_{\substack{0\le i\le N \\ 1\le k\le n_i \\ 1\le j\le n_{i+1}}}$ is the matrix of the weights,

N is the number of hidden layers, and n_i is the number of neurons by the n^{th} hidden layers.
Variables that can be summed up search in the following matrix:

$$U = \left[u_1,...,u_i,...,u_{N-1}\right]$$

$$W = \left[W^0,...,W^i,...,W^N\right]$$

where $W^i = \left(w_{k,j}^{i}\right)_{\substack{1\le k\le n_i \\ 1\le j\le n_{i+1}}} \quad i = 0,...,N$

Remark
The optimal number of hidden layers is defined as follows:

$$n_{opt} = \sum_{i=1}^{N-1} u_i$$

Our approach begins with a maximal number of hidden layers. The training process, all observations are sequentially input to learning system in an arbitrary order. The number of hidden layers and the values of weights are needed by the optimization model. In addition, the number of hidden layers must be decided simultaneously with a training phase (weights adjust).

Example
Let a multi-layer neural network with two hidden layers, In this example, the number of hidden layers is equal to 2, the number of neurons by each hidden layers is equal to3.
The rule of the variable u_i can be explained by the following example:

Size of the observations set	10
Dimension of desired space	2
Dimension of input	3
Number max of hidden layers	3
Activation function	f(x)=x
$n_0, n_2, ,n_3$	3
n_1	4
n_4	2

Remark
The set of observation is generated randomly.

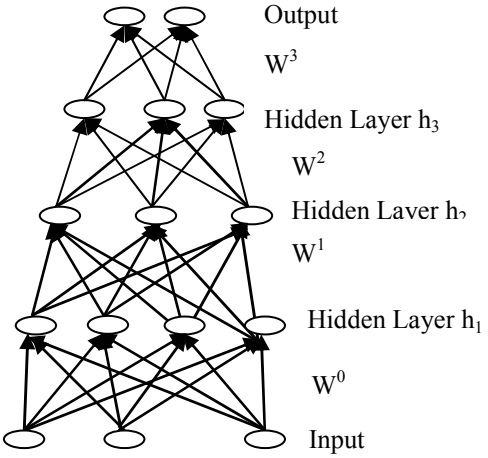

Fig 5: Example for the neural architecture optimization

The model corresponds to this example is defined as follows:

$$(P)\begin{cases} Min\left\|F(U,W,X)-d\right\|^2 \\ Sc: \\ \\ u_1+u_2 \ge 1 \\ \\ W = \left(w_{k,j}^{i}\right)_{\substack{0\le i\le 3 \\ 1\le k\le n_i \\ 1\le j\le n_{i+1}}} \quad w_{k,j}^{i} \in IR \\ u_1, u_2 \in \{0,1\} \end{cases}$$

where

$W = \left(w_{k,j}^{i}\right)_{\substack{0\le i\le 3 \\ 1\le k\le n_i \\ 1\le j\le n_{i+1}}}$,$i = 0,...,3$ is the matrix of the weights.
The transfer function is defined as:

$$F(W,U,X) = \begin{pmatrix} f\left(\sum_{k=1}^{3} w_{k,1}^{3} h_3^k\right) \\ \\ f\left(\sum_{k=1}^{3} w_{k,2}^{3} h_3^k\right) \end{pmatrix}$$

h_3^i is the output for the i[th] neuron by the 2[th] hidden layer.

$$h_3 = \begin{pmatrix} h_3^1 \\ h_3^2 \\ h_3^3 \end{pmatrix} = (1-u_2)h_1 + u_2 \begin{pmatrix} f\left(\sum_{k=1}^{3} w_{k,1}^2\ h_2^k\right) \\ f\left(\sum_{k=1}^{3} w_{k,2}^2\ h_2^k\right) \\ f\left(\sum_{k=1}^{3} w_{k,3}^2\ h_2^k\right) \end{pmatrix},$$

$$h_2 = \begin{pmatrix} h_2^1 \\ h_2^2 \\ h_2^3 \end{pmatrix} = (1-u_1)h_1 + u_1 \begin{pmatrix} f\left(\sum_{k=1}^{3} w_{k,1}^1\ h_1^k\right) \\ f\left(\sum_{k=1}^{3} w_{k,2}^1\ h_1^k\right) \\ f\left(\sum_{k=1}^{3} w_{k,3}^1\ h_1^k\right) \end{pmatrix},$$

$$h_1 = \begin{pmatrix} h_1^1 \\ h_1^2 \\ h_1^3 \end{pmatrix} = \begin{pmatrix} f\left(\sum_{k=1}^{3} w_{k,1}^0\ x_k\right) \\ f\left(\sum_{k=1}^{3} w_{k,2}^0\ x_k\right) \\ f\left(\sum_{k=1}^{3} w_{k,3}^0\ x_k\right) \end{pmatrix},$$

Remark
The i[th] hidden layer is deleted, the $(i\text{-}1)^{th}$ hidden layer is connected to the $(i+1)^{th}$ one.

In this example, we have four cases:
Case 1:
u_1=1 and u_2=1
Case 2:
u_1=1 and u_2=0
Case 3:
u_1=0 and u_2=1
Case 4:
u_1=0 and u_2=0

We studying each case, we obtained the optimal solution, where we have u1=0 and u_2=1. Consequently, the first hidden layer is deleted, and the input is connected to the 2[th] hidden layer.

The optimal architecture is defined as follows:

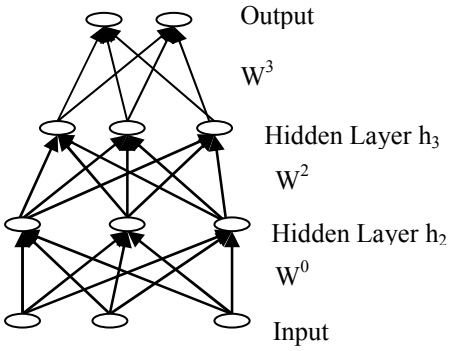

Fig 6: Example for the neural architecture optimization

4. Solving the obtained non linear programming model

Genetic algorithm is proposed to solve this problem. Genetic Algorithm belongs to a class of probabilistic methods called "evolutionary algorithms" based on the principles selection and mutation. GA was introduced by J. HOLLAND [13] that it's based on natural evolution theory of Darwin. Each solution represents an individual who is coded in one or several chromosomes. These chromosomes represent the problem's variables. First, an initial population composed by a fix number of individuals is generated, then, operators of reproduction are applied to a number of individuals selected switch their fitness. This procedure is repeated until the maximums number of iterations is attained. G.A. has been applied in a large number of optimization's problems in several domains [6], telecommunication, routing, scheduling, and it proves it's efficiently to obtain a good solutions [2]. We have formulated the problem as a non linear program with mixed variables.

Genetic Algorithm's framework:

1. Coding individuals
2. Generate the initial population
3. Repeat
   - Evaluation individuals
   - Selection of individuals
   - Crossover
   - Mutation
   - Reproduction

   Until attaining the criteria

Coding

In our application, we have encoded an individual by two chromosomes, the first one represent the matrix of weights "W", and the second represents the vector "*u*" which is an array of decision variables who takes 1 if the hidden layer is activated and 0 other.

Example:

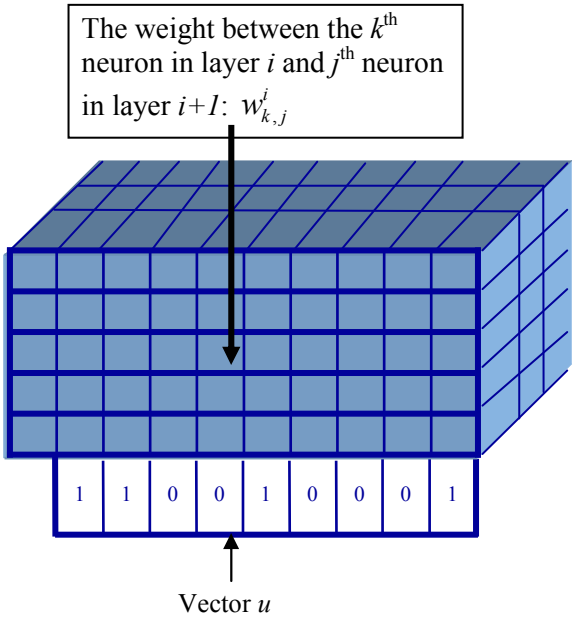

Fig 7: Coding

Initial Population

Individuals of the initial population are randomly generated, where vector "*u*" variables take the value 0 or 1, and the matrix of weights takes random values in space IR.

There are other ways to generate the initial population like applying others heuristics, this is the case when the research space is constrained what is not our case.

Evaluating individuals

In this step, each individual is assigned a numerical value called fitness which corresponds to its performance; it depends essentially on the value of objective function in this individual. An individual who has a great fitness is the one who is the most adapted to the problem.

The fitness suggested in our work is the following function:

$$Fitness(i) = M - objective(i)$$

where $M>>0$.

Minimize the value of the objective function "objective" is equivalent to maximize the value of the fitness function.

Selection

The selection method used in this paper is the Roulette Wheel Selection RWS, which is a proportional method of selection, in this method; individuals are selected according to their fitness. The principle of RWS can be summarized in the following schema:

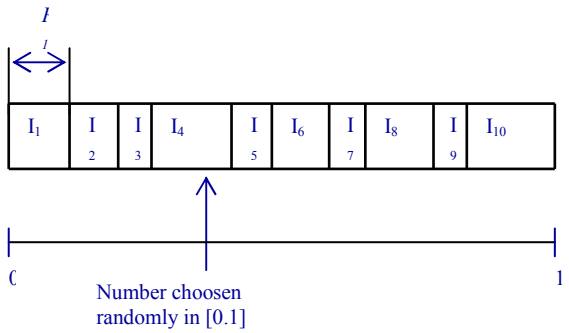

Fig 8: Selection RWS

where

$$P_i = \frac{f_i}{\sum_{i=1}^{n} f_i}$$

Crossover

The crossover is very important in the algorithm, in this step, new individuals called children are created by individuals selected from the population called parents. Children are constructed as follows:

We fix a point of crossover, the parent are cut switch this point, the first part of parent 1 and the second of parent 2 go to child 1 and the rest go to child 2.

In the crossover that we adopted, we choose two different crossover points, the first for the matrix of weights and the second is for Vector *U*.

Example:

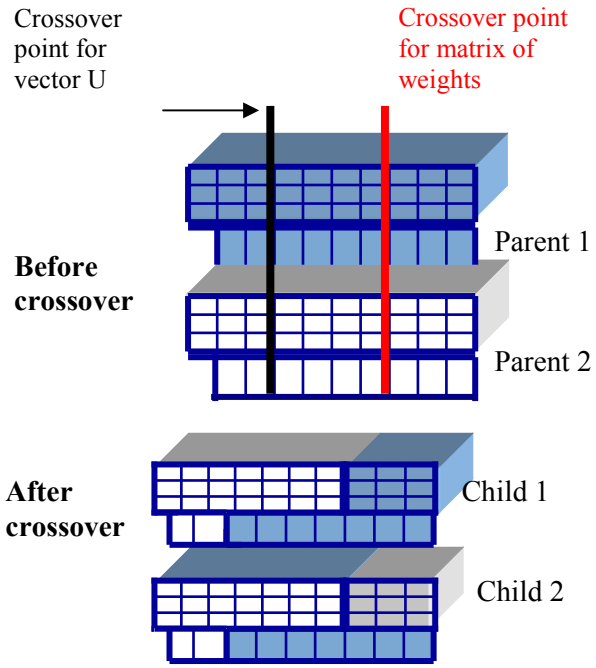

Fig 9: Crossover

Mutation

The role of mutation is to keep the diversity of solutions in order to avoid local optimums. It corresponds on changing the values of one (or several) value (s) of the individuals who are (or were) (s) chosen randomly.

Example:

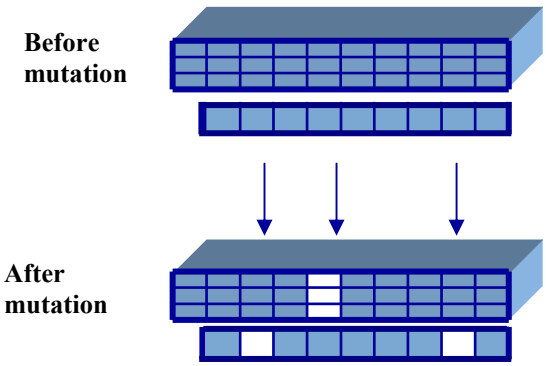

Fig 9: Mutation

Algorithm

The scheme of the algorithm is depicted in the following figure.

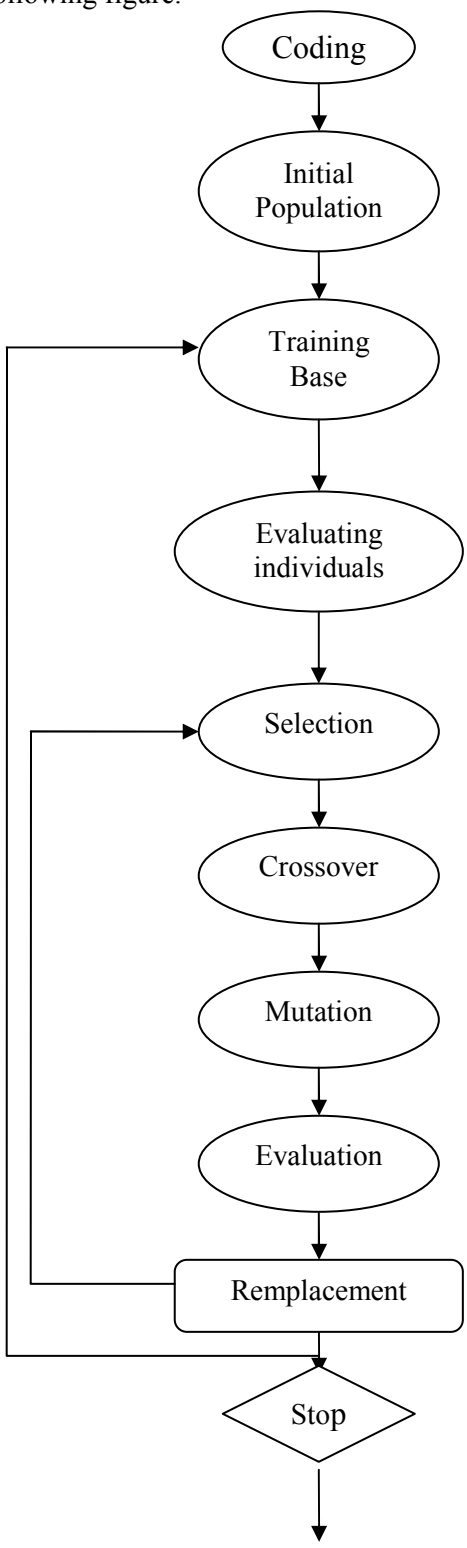

Fig 10: Algorithm description

5. Experiments Results

In this section, we provide the numerical examples to illustrate the potential applications of the proposed methodology.

The architecture of multi layer network is used in this paper.

The activation function is used in this architecture is given as:

$$f(x) = \left(1 + \exp(x)\right)^{-1}$$

Performance evaluation

Two different types of standard statistical were considered as statistical performance evaluation. The mean prediction error is calculated by the root mean squared error (RMSE) and mean percentage error (MAPE) were used. The performance evaluation criteria used in this article can be computed using the following expressions:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{Q} \left\| Y^i - d^i \right\|^2}{Q}}$$

$$MAPE = \frac{1}{Q} \sum_{i=1}^{Q} \frac{\left\| Y^i - d^i \right\|}{\left\| Y^i \right\|} \times 100$$

Where, Y^i denotes the actual (calculated) value, d the predicted value, where the i^{th} observation is counted from 1 to Q and Q is the total number of observations.

The experimental settings for neural architecture optimization shows in table 1, in order to study some of the properties of data used:

Population size	10
Size of learning set	10
Dimension of desired space	15
Dimension of input	10
Number max of hidden layers	10
Number of neurons by each hidden layer	10

Table 1: experimental settings for neural architecture

In order to test all theoretical results introduced in this paper, a computer program has been designed. Our algorithm was implanted in C++, and tested on instances generated randomly. The program was run in a compatible IBM, Pentium III 666MHz and 128 Mb of RAM. The experimental results are presented in the table 2:

N. Max of iteration	N	n_{opt}	RMSE	MAPE (%)	Number of iteration	CPU(s)
100	6	4	1.41	24.56	19	2.25
	6	2	1.53	23.212	25	2.20
	8	5	0.86	17.20	96	3.08
	8	4	0.85	30.77	73	3.08
	10	3	1.25	13.75	55	3.96
	10	7	1.28	21.28	37	4.12
1000	6	4	1.08	19.02	213	22.58
	8	5	1.02	15.21	627	31.55
	8	2	1.16	20.35	613	31.26
	10	4	0.87	12.99	752	48.30
	10	7	0.91	14	915	49.45
10000	6	5	0.90	15.19	9816	247.42
	6	5	0.83	13.78	1432	247.4
	8	4	0.73	11.88	9433	344.12
	10	4	0.78	12.97	9043	481.70

Table 2: Results of neural architecture optimization

In Table 2, we present the obtained number of hidden layers and the performance criteria, we analyse the performance of our method for different observations witch are generated randomly.
In the following graphics, we interpret the obtained results for the problem of the neural architecture optimization.

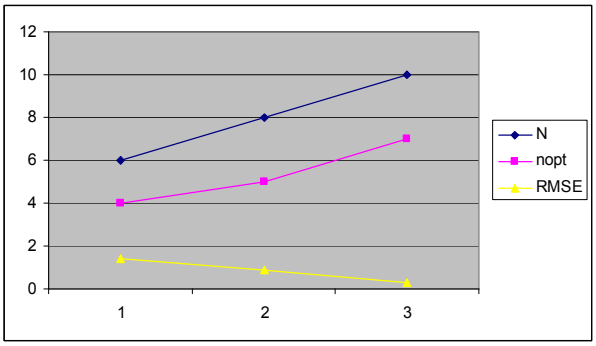

Fig 11: Optimal solution for 100 iterations

The numerical results assess the effectiveness of the theorical results shown in this paper, and the advantages of the new modelling.

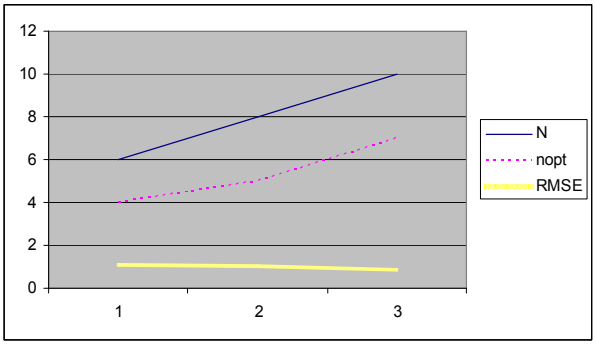

Fig 12: Optimal solution for 1000 iterations

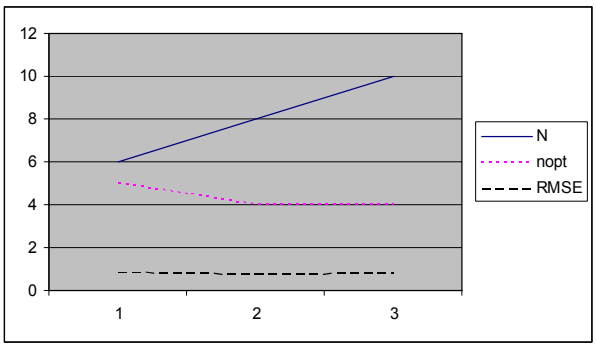

Fig 13: Optimal solution for 10000 iterations

6. Conclusion

ANN is widely accepted as a technology offering an alternative way to simulate complex and ill-defined problems. They have been used in diverse applications in control, robotics, pattern recognition, forecasting power systems, manufacturing, optimization, signal processing,…
In this paper, we have proposed a new modelling for the neural architecture optimization problem. The GA is especially appropriate to obtain the optimization solution of the complex nonlinear problem. This method is tested to determine the optimal number of hidden layers in the ANN and the most favourable weights matrix, in order to minimize the error between the desired (predicted) output and the calculated output. Determining these parameters has an impact on the performance of ANN. Several directions can be investigated to try improving this method use a more efficient metaheuristics for example: tabu research, ant colony system [6]. Clearly, heuristical and mathematical approaches using a priori information would give rise to better performance and stability.
To make this approach more efficient, it can be combined with other metaheuristics or it can be computationally optimised by introducing analytical improvements, such as replacing the hyperbolic tangent, with a linear function.
Moreover, the approach introduced in this paper is also intrinsically easy to parallelize.
Other experiment results are in progress for searching adequate values of parameters such as threshold value of the GA, Tabu research, and Ant colony system.
Others studies are in progress to apply this approach to many problems such as: image processing, classification, image filtration [10], optimization.

Acknowledgements
The authors would like to express their sincere thanks for the referee for his/her helpful suggestions.

References

[1] M. K. Apalak, M. Yildirim, R Ekici, Layer optimization for maximum fundamental frequency of laminated composite plates for different edge conditions, *composites science and technology* (68) 537-550, 2008.

[2] Bouktir, T., Slimani, L., Optimal power flow of the Algerian Electrical Network using genetic algorithms, WSEAS TRANSACTIONS on

CIRCUITS and SYSTEMS, Vol. 3, Issue 6, 2004, pp. 1478-1482.

[3] INCI CABAR, SIRMA YAVUZ2, OSMAN EROL1, 'Robot Mapping and Map Optimization Using Genetic Algorithms and Artificial Neural Networks', WSEAS TRANSACTIONS on COMPUTERS, Issue 7, Volume 7, July 2008.

[4] A. Cichoki, R. Unberhuen, 'Neural Network for Optimization and signal processing', *Jhon Wiley & Sons*, New York, 1993.

[5] Y. Chang, C. Low, 'Optimization of a passive harmonic filter based on the neural-genetic algorithm with Fuzzy logic for a steel manufacturing plant', *Expert systems with Application*, 2059-2070,2008.

[6] J Dréo, A. Pétrowski, P. Siarry, É. Taillard, 'Méta-heuristiques pour l'optimisation difficile' , *Éditions Eyrolles*, 2003.

[7] G. Dreyfus, J. Martinez, M. Samuelides, M. B. Gordon, F. Badran, S.Thiria, L.Hérault, 'Réseaux de neurones. Méthodologie et applications', *Edition Eyrolles 2e édition, 2003.*

[8] E. Egriogglu, C, Hakam Aladag, S. Gunay, 'A new model selection straegy in artificiel neural networks', *Applied Mathematics and Computation* (195) 591-597, 2008.

[9] M. Ettaouil, 'Contribution à l'étude des problèmes de satisfaction de contraintes et à la programmation quadratiques en nombre entiers, allocation statiques de tâches dans les systèmes distribués', *thèse d'état, Université Sidi Mohammed ben Abdellah, F.S.T. de Fès*, 1999.

[10] M. Ettaouil, Y. Ghanou, 'Réseaux de neurones artificiels pour la compression et la filtration de l'image médicale', *congres de la société marocaine de mathématiques appliquées, proceedings* pages 148-150 Rabat 2008.

[11] M. Ettaouil, Y Ghanou, 'Méta- heuristiques et optimisation des architectures neuronales', *JIMD'2 procedings ENSIAS Rabat*, 2008.

[12] M. Ettaouil and C. Loqman, 'Constraint satisfaction problem solved by semi definite relaxation', WESAS TRASACTIONS ON COMPUTER, Issue 7, Volume 7, 951-961, 2008.

[13] J. Holland, 'Adaptation in natural and artificial systems'. Ann Arbor, MI: *University of Michigan Press*, 1992.

[14] Julio J. Valdes, Alan J. Barton1,' Multi-objective evolutionary optimization for constructing neural networks for virtual reality visual data mining: Application to geophysical prospecting', *Neural Networks* 20 (2007) 498–508.

[15] T. Kohonen, 'Self Organizing Maps', *Springer*, 3e edition, 2001.

[16] T.Y. Kwok, D.K. Yeung, 'Constructive algorithms for structure learning in feed forward neural networks for regression problems', *IEEE Trans. Neural Networks* 8 (1997) 630-645.

[17] CHE-CHERN LIN, 'Implementation Feasibility of Convex Recursive Deletion Regions Using Multi-Layer Perceptrons', WSEAS TRANSACTIONS on COMPUTERS, Issue 1, Volume 7, January 2008.

[18] H Peng, X Ling, 'Optimal design approach for the plate-fin heat exchangers using neural networks cooperated with genetic algorithms', *Applied Thermal Engineering* 28 (2008) 642–650.

[19] JOSEPH RAJ V. 'Better Learning of Supervised Neural Networks Based on Functional Graph – An Experimental Approach', WSEAS TRANSACTIONS on COMPUTERS, Issue 8, Volume 7, August 2008.

[20] Pedro M. Talavan, J. Yanez, 'A continuous Hopfield network equilibrium points algorithm', *Computers & Operations Research* 32 (2005) 2179–2196

[21] Z. Tang, C.Almeida, P.A Fishwick, 'Time series forecostng using neural network vs Box-jenkins methodology', *Simulation* 57 (1991)303-310.

[22] D. Wang, 'Fast Constructive-Coverting Algorithm for neural networks and its implement in classification', *Applied Soft Computing* 8 (2008) 166-173.

[23] D. Wang, N.S. Chaudhari, 'A constructive unsupervised learning algorithm for Boolean neural networks based on multi-level geometrical expansion', *Neurocomputing* 57C (2004) 455-461.